

Lecture 4 Date, Time Objects, Return and Plot

Ziwen Ye

Stevens Institute of Technology

zye2@stevens.edu

September 22, 2018

- 1 Date class in R
 - Generate date
 - Change Date format in R
- 2 POSIXt class
 - as.POSIXct and as.POSIXlt
- 3 Return
 - Simple return
 - Log return
- 4 Basics
- 5 Different type of plot
- 6 Other advanced settings

Why we need to learn Date class

The core data object for holding data in R is the *data.frame object*. A *data.frame* is a rectangular data object whose columns can be of different types (e.g., numeric, character, logical, Date, etc.). The *data.frame* object, however, is not designed to work efficiently with time series data. In particular, subsetting and merging data based on a time index is cumbersome and transforming and aggregating data based on a time index is not at all straightforward. Furthermore, the default plotting methods in R are not designed for handling time series data. Hence, there is a need for a flexible time series class in R with a rich set of methods for manipulating and plotting time series data.

as.Date is a functions to convert between character representations and objects of class "Date" representing calendar dates.

Example

```
> myDate <- as.Date("1970-01-01")
> mode(myDate)
[1] "numeric"
> # [1] "numeric"
> class(myDate)
[1] "Date"
> # [1] "Date"
> # internally it is number of days since Jan 01, 1970
> as.numeric(myDate)
[1] 0
> # [1] 0
```

In *as.Date* it only support 1 single object rather than 2

Example

```
> #Bad example
> myDates <- as.Date("2013-01-01", "2013-01-02")
> myDates
[1] NA
> # doesn't work with 2 variables, need to pass a vector
>
> #Good example
> myDates <- as.Date(c("2013-01-01", "2013-01-02"))
> as.numeric(myDates)
[1] 15706 15707
```

In *as.Date* it can only recognize some standard time type, other type of time it may not able to recognize.

Example

```
as.Date("1970/01/01")
# [1] "1970-01-01"
as.Date("1970-01-01")
# [1] "1970-01-01"
as.Date("01/01/1970") #non-standard input
# [1] "0001-01-19"

# set up time format when generate time data
as.Date("1970-01-01", format = "%Y-%m-%d")
# [1] "1970-01-01"
as.Date("02/24/2015", format = "%m/%d/%Y")
# [1] "2015-02-24"
```

Format codes for dates

Code	Value	Example
%d	Day of the month	23
%m	Month	01
%b	Month(abbreviated)	Jan
%B	Month(full)	January
%y	Year(2 digits)	90
%Y	Year(4 digits)	1990

When change date type, you can use *format()*.

Other useful function

- `months()`
- `quarters()`
- `weekdays()`
- `seq.Date()`
- `seq()`

Introduction

There are two POSIXt subclasses available in R: POSIXct and POSIXlt. The POSIXct class represents datetime values as the signed number of seconds (which includes fractional seconds) since midnight GMT (UTC universal time, coordinated) 19700101. This is analogous to the Date class with addition of times during the day. The POSIXlt class represents datetime values as a named list with elements for the second (sec), minute (min), hour (hour), day of the month (mday), month (mon), year (year), day of the week (wday), day of the year (yday), and daylight savings time flag (isdst), respectively.

Main difference from POSIXct and POSIXlt

Example

```
myDateTime1 <- as.POSIXct("01,01,2014 10:20:20",  
                           format = "%d,%m,%Y %H:%M:%S")  
  
myDateTime2 <- as.POSIXlt("01,01,2014 10:20:20",  
                           format = "%d,%m,%Y %H:%M:%S")  
  
myDateTime1$sec  
# Error: object 'myDatetime1' not found  
myDateTime2$sec  
# [1] 20
```

strptime() and difftime()

Example

```
> # strptime()
> dt1 <- strptime("2013.12/23 01:00:34", "%Y.%m/%d %H:%M:%S")
> dt2 <- strptime("2013.12/23 01:02:37", "%Y.%m/%d %H:%M:%S")
> dt1
[1] "2013-12-23 01:00:34 EST"
> dt2$sec
[1] 37
>
> #You get time difference directly by using "-"
> dt2-dt1
Time difference of 2.05 mins
> #use difftime can change unit
> difftime(dt2, dt1, units = "secs")
Time difference of 123 secs
```

Brief summary

- `strptime()` is the most basic function, it only accept character as input. This is why this function is much faster in dealing with large input.
- `strptime()` and `as.POSIXlt()` returns a list
- `as.POSIXct()` returns a string
- Be extra careful when you set up the time zone

Why use return series instead of price series.

- Return is a complete and scale-free summary of an asset.
- Return has more attractive statistical properties. (weakly stationary)

Simple Return

$$1 + R_t = \frac{P_t}{P_{t-1}}$$

$$\therefore R_t = \frac{P_t}{P_{t-1}} - 1 = \frac{P_t - P_{t-1}}{P_{t-1}}$$

Log Return (continuously compounded return)

$$r_t = \log(1 + R_t) = \log(P_t/P_{t-1})$$

$$= \log(P_t) - \log(P_{t-1})$$

Return

For multiple time intervals, say K:

Simple

$$\begin{aligned}1 + R_{t[k]} &= \frac{P_t}{P_{t-k}} = \frac{P_t}{P_{t-1}} \frac{P_{t-1}}{P_{t-2}} \cdots \frac{P_{t-k+1}}{P_{t-k}} \\ &= (1 + R_t)(1 + R_{t-1})(1 + R_{t-2})\end{aligned}$$

Log

$$\begin{aligned}r_{t[k]} &= \log(1 + R_{t[k]}) = \log\left[\frac{P_t}{P_{t-1}} \frac{P_{t-1}}{P_{t-2}} \cdots \frac{P_{t-k+1}}{P_{t-k}}\right] \\ &= \log\left(\frac{P_t}{P_{t-1}}\right) + \cdots + \log\left(\frac{P_{t-k+1}}{P_{t-k}}\right) = \sum_{j=1}^{k-1} r_j\end{aligned}$$

Continuously Compounding

$$\text{one year: } \lim_{m \rightarrow \infty} \left(1 + \frac{r}{m}\right)^m = e^r$$

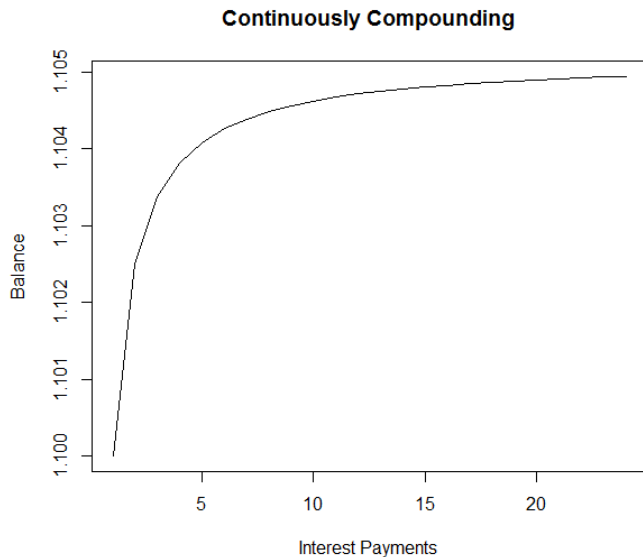
$$t \text{ years: } \lim_{m \rightarrow \infty} \left(1 + \frac{r}{m}\right)^m{}^t = e^{rt}$$

Continuously Compounding

Example

```
> result <- NULL
> x <- 1:24
> for (m in x)
+ {
+   result <- c(result, contCompound(0.1, m))
+ }
> plot.ts(result, main = "Continuously Compounding",
           xlab = "Interest Payments", ylab = "Balance")
```

Continuously Compounding



plot()

plot() is the most basic graphic function in R, it doesn't need any package to start. This function is able to generate line scatter plot, line graph, bar graph, histogram and box plot.

```
plot()
```

```
plot(x, y, ...)
```

x means inputs in x-axis

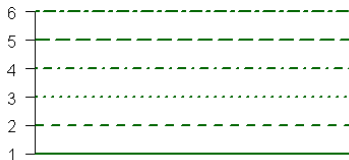
y means inputs in y-axis

Advanced arguments

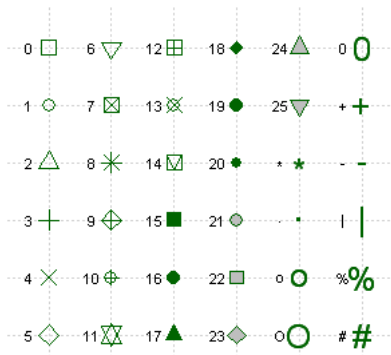
In `plot()` function, user is able to adjust plot format with following arguments.

Argument	Description
<code>main</code>	title for the plot
<code>xlab,ylab</code>	label for x or y axis
<code>xlim,ylim</code>	value range on x or y axis
<code>lwd</code>	line width
<code>lty</code>	type of line
<code>pch</code>	type of point
<code>col</code>	color for line or point

Line Types: lty=



plot symbols : pch =



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125
126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150
151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200
201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225
226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250
251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275
276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300
301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325
326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350
351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375
376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400
401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425
426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450
451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475
476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500
501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525
526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550
551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575
576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600
601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625
626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650
651	652	653	654	655	656	657																		

Also you can refer to this

<http://www.stat.columbia.edu/tzheng/files/Rcolor.pdf>

Scatter plot

Scatter plot is a plot use coordinate to describe relationship between two variable

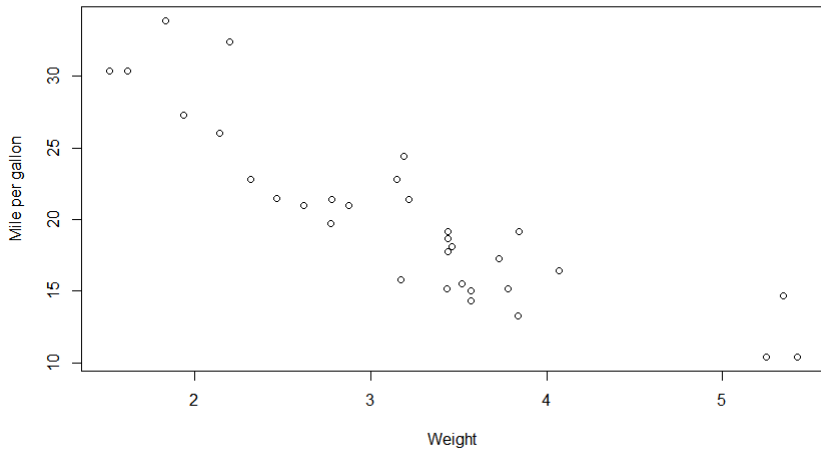
Example

```
plot(mtcars$wt, mtcars$mpg)
```

Advanced setting

1. Adding labels and title
2. Change point type

Scatter Plot



Bar plot

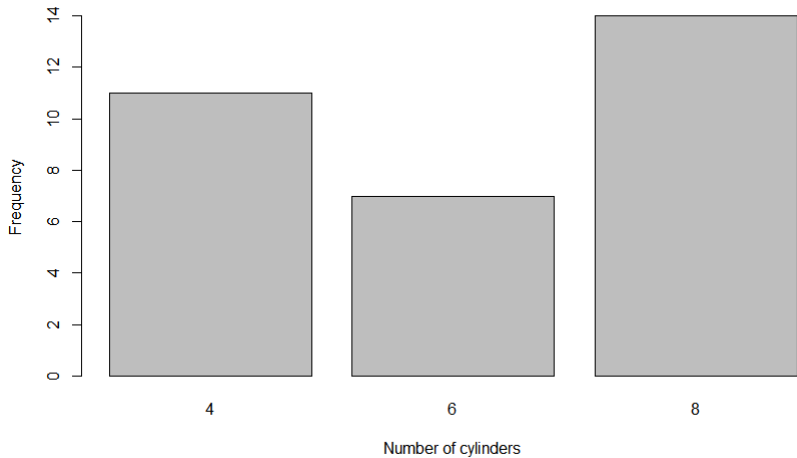
Bar plot can represent grouped data with rectangular bars

Example

```
barplot(table(mtcars$cyl))
```

Advanced setting

1. Change color of bar
2. Create stacked bar plot



Legend

Legend is an important part for any plot, this is the place to identify your plot content.

Example

```
legend("location", names, fill=cols)
```

In most case, location, names and cols are the most basic information to put in legend.

For detailed information please check code.

Box plot

In descriptive statistics, a box plot is a convenient way of graphically depicting groups of numerical data through their quartiles.

Example

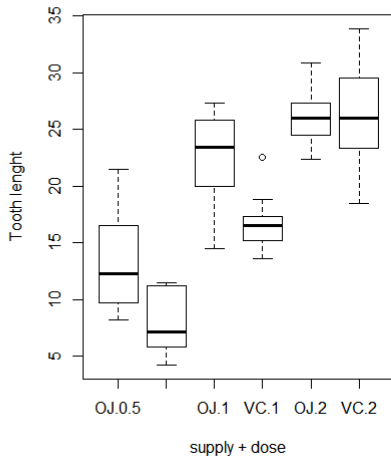
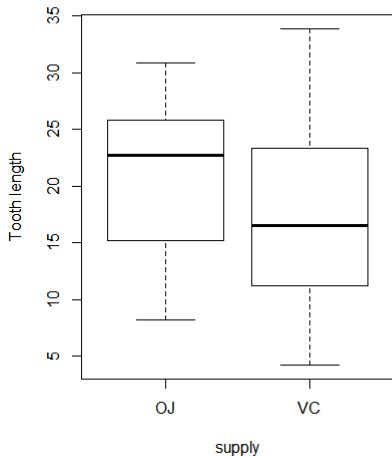
```
boxplot(len ~ supp + dose, data = ToothGrowth)
```

Advanced setting

1. color for border and bar
2. point style for outlier
3. change median number color

If you want to re-arrange the location of the plot, you need to refer `par()` or `layout()`. No matter which one you are using, it will relocate the plot by "matrix format". However, `layout()` is more flexible.

- `par()` example is shown in the box plot section
- `layout()` example is shown in the histogram section



Box plot summary

- The format of box plot is `boxplot(Y value X value)`
- The input of Y-value can only be **numeric** vector
- The input of X-value do not have limitation (in most case, it is a factor vector)

Rewrite description for axes

To rewrite description for axes, we need to delete original content before we generate plot, or new description will overlap with old content.

Delete description

```
xaxt = "n"  
yaxt = "n"
```

Above function can not be used individually, they are arguments in `plot()` function.

Change description for line chart

Sometime the line chart can not present a ideal output for time objects. In such case, removing the description from corresponding axis is the best solution.

For detailed information please refer to code.

Key word

```
axis()  
axis.POSIXct()
```